Modularizing a Java app using Java 9

Sabari Krishnamoorthy
OpenJ9 Cloud Squad,
IBM Runtime Technologies

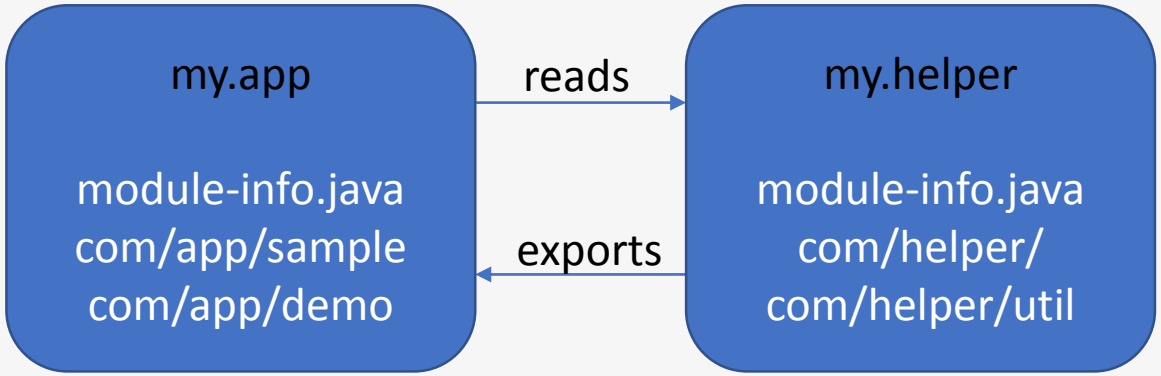
Agenda

- Quick peek into modularity
- What is AcmeAir?
- Applying modularity to AcmeAir

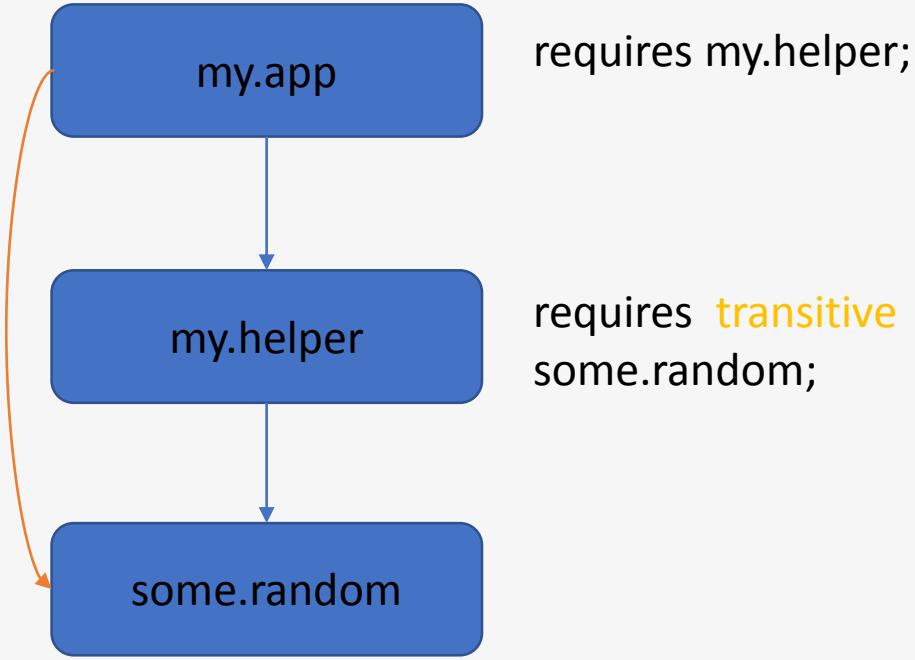
Quick peek into modularity

- What is a module?

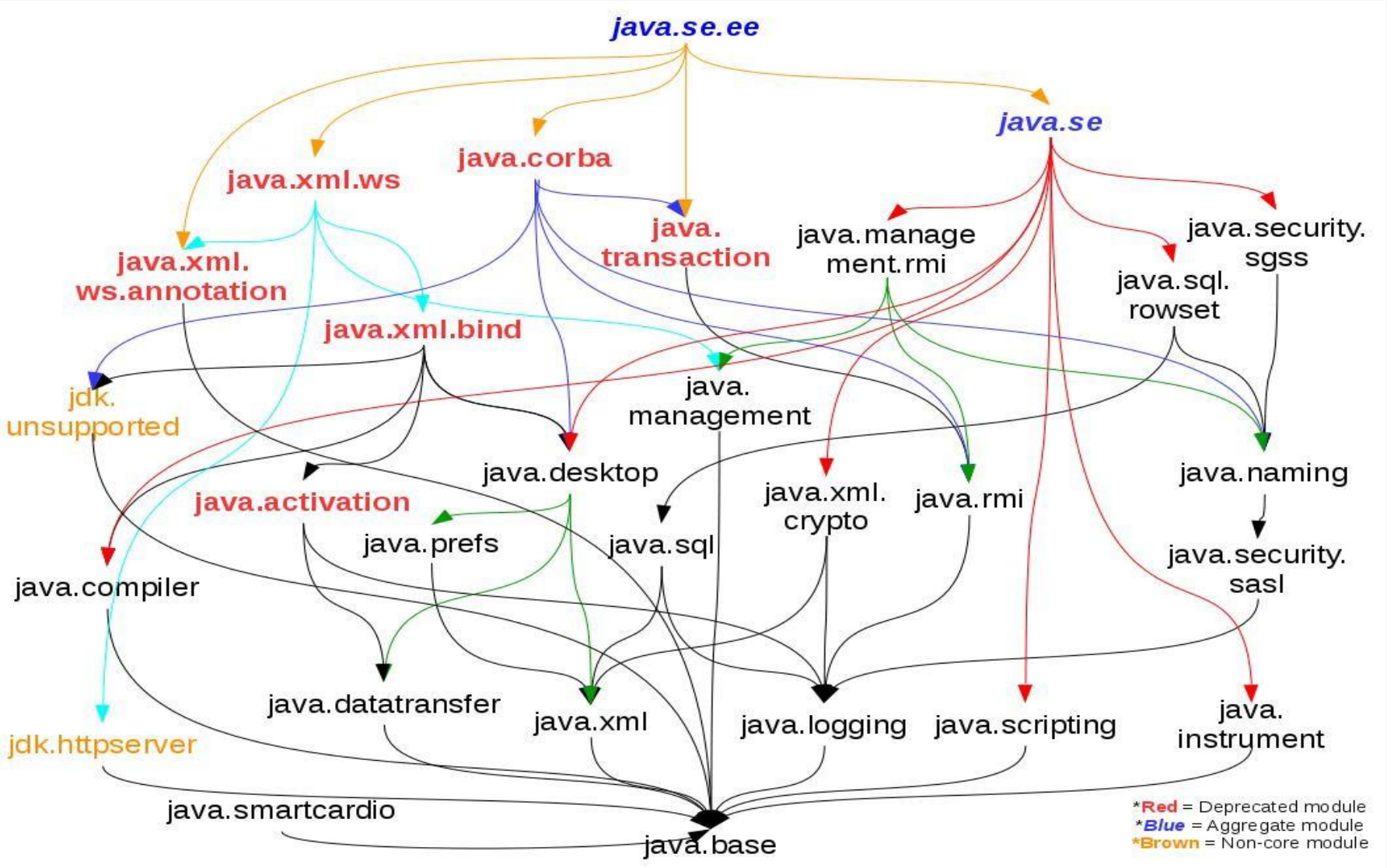
Accessibility



- Implied-readability



Quick peek into modularity (Cont)



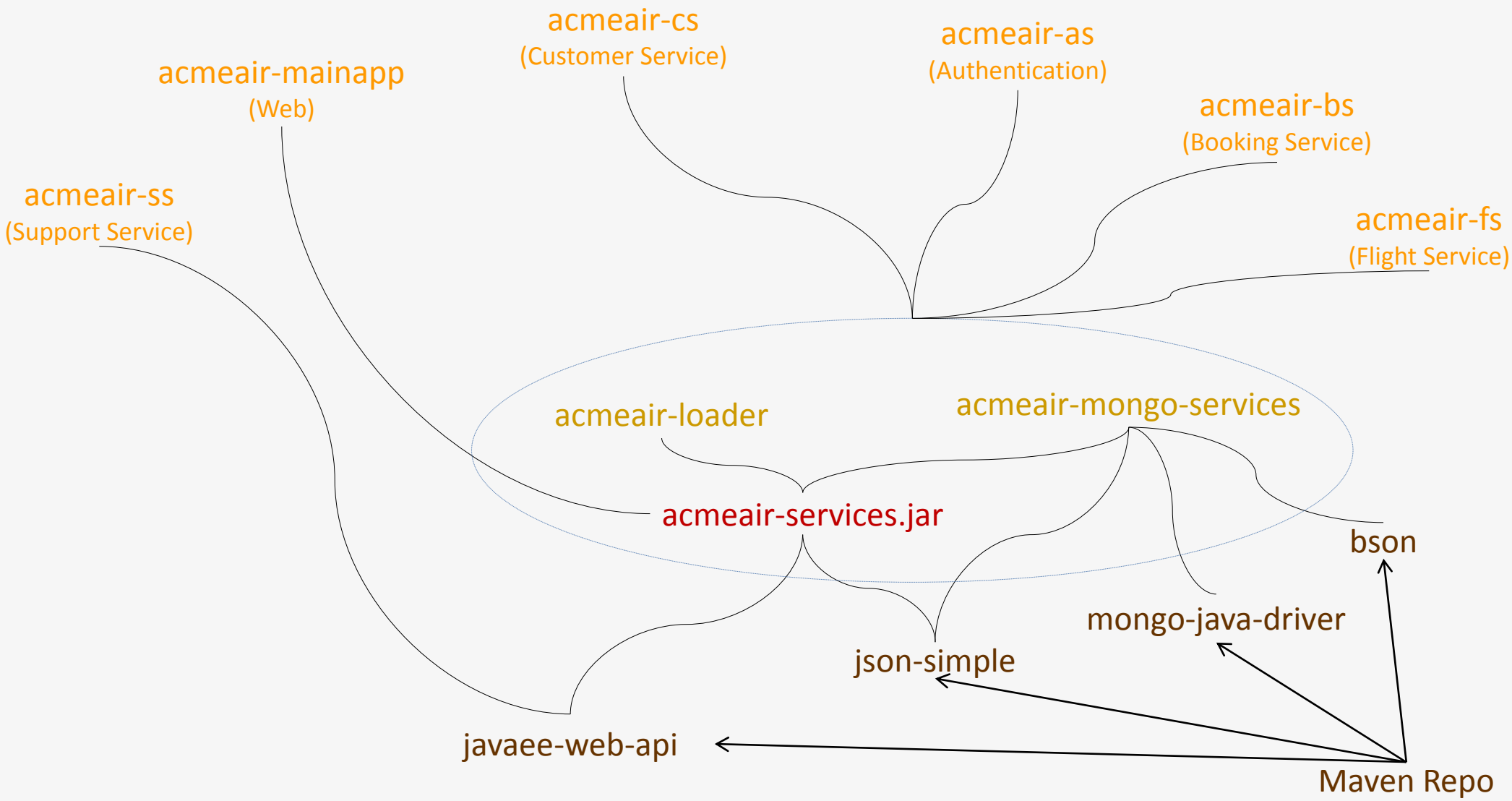
Quick peek into modularity (Cont)

- Types of modules
  - Explicit
  - Automatic
  - Unnamed
- jdeps tool
  - -jdkinternals
  - dependency analysis
  - generate module info

What is AcmeAir?

- Performance benchmark
- Built with key business requirements
  - Ability to scale to billions of web API calls per day
  - Need to develop and deploy the application in clouds (public, private and hybrid)
  - Need to support multiple channels for user interaction
- Available as both monolithic and microservices

AcmeAir – Non-modular layout



Demo

(https://github.com/ibmruntimes/acmeair-modular)

Related Links

Related JEPs and JSRs

- http://openjdk.java.net/projects/jigsaw/
  - 200: The Modular JDK → http://openjdk.java.net/jeps/200
  - 201: The modular source code → http://openjdk.java.net/jeps/201
  - 220: The modular Run-Time images → http://openjdk.java.net/jeps/220
  - 260: Encapsulate most internal APIs → http://openjdk.java.net/jeps/260
  - 261: Module System → http://openjdk.java.net/jeps/261
  - 282: jlink: The java linker → http://openjdk.java.net/jeps/282

- Gradle modularity changes
  - https://guides.gradle.org/building-java-9-modules/
  - https://discuss.gradle.org/t/modularity-support-with-gradle/22445